ДОДАТОК А

Лістинг А.1 – VHDL-модель кодера

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

-- Custom package
library xil_defaultlib;
   use xil_defaultlib.f_files_encoder.all;
entity encoder is
   generic(
      InfowordLength : integer := 64;
      CodewordLength : integer := 128
      );
   port(
        input_data   : in std_logic_vector(0 to InfowordLength -
1);
     input_clk    : in std_logic;
     clk_not      : in std_logic;
      ---------------------------
       output_data  : out std_logic_vector(0 to CodewordLength -
     1);
      valid        : out std_logic
      );
end encoder;

architecture encoder_bh of encoder is
----------- Components ------------------------------------
   component shift_register is
      generic(
         WIDTH : integer := 64
         );
          port(
         clk : in STD_LOGIC;
         load : in STD_LOGIC;
         load_vector : in STD_LOGIC_VECTOR (0 to WIDTH - 1);
         en    : in std_logic;
         -------------------------------
         Output : out STD_LOGIC_VECTOR(0 to WIDTH - 1)
         );
   end component shift_register;
   --------------------
   component BUFG
      port(I: in STD_LOGIC; O: out STD_LOGIC);
   end component;
```

```
   ----------- Constants -----------------------------------
    constant m : integer := (CodewordLength - InfowordLength) /
4;
                                                    -- 16
   ----------- Shift memory --------------------------------
   signal shift_load_vector : STD_LOGIC_VECTOR(0 to m * 4 - 1);
   signal shift_output      : STD_LOGIC_VECTOR(0 to m * 4 - 1);
   signal shift_en          : std_logic := '0';
   signal shift_load        : std_logic := '0';

   ----------- States --------------------------------------
   type STATES is (STATE_Wait, STATE_Load_mem, STATE_Shifting);
   signal state: STATES := STATE_Wait;

   ----------- Output temp ---------------------------------
    signal output_t : std_logic_vector (0 to m * 4 - 1) :=
                                              (others => '0');
   ------------------------------
   signal i : integer := 0;    -- 0 to 3
   signal j : integer := 0;    -- 0 to 63
   signal iteration : integer := 0;
   signal input : std_logic_vector (0 to InfowordLength - 1);
    --signal shift_output : std_logic_vector (0 to 64 - 1) :=
                                              (others => '0');
   signal clk : std_logic;

begin
     shift_memory: shift_register generic map(CodewordLength -
InfowordLength)
      port map(clk, shift_load, shift_load_vector, shift_en,
shift_output);
   BUFG_mark : BUFG port map(clk_not, clk);

   process (clk)
      begin
      if rising_edge(clk) then
        case (state) is

            when STATE_Wait =>
             if (input_clk = '1') then
                valid <= '0';
              input <= input_data;
              -- Loading shift memory --
              for i in 1 to 4 loop
                  shift_output ((i-1)*m to i*m - 1) <= row1(i*m
                                - 1) & row1((i-1)*m to i*m - 1 -
1);
              end loop;

              shift_load <= '1'; -- Control signal for load
                                                      shift
memory
```

```
              if (input_data(0) = '1') then -- Adding process,
                            wher (i-1)*m + j - 1 is number of
iteration
                 output_t <= row1;
              else
                 output_t <= (others => '0');
              end if;
              iteration <= 1;
              i <= 0;
              j <= 1;
              state <= STATE_Shifting;
           else
        end if;


           when STATE_Shifting => -- 512 shifts of shift memory
                                                  and 512 sum
operations
            shift_load <= '0'; -- Control signal for load shift
                                                       memory
              shift_en <= '1';

            if (input(iteration) = '1') then -- Adding process,
                            wher (i-1)*m + j - 1 is number of
iteration
               output_t <= output_t xor shift_output;
            else
            end if;

            for i in 1 to 4 loop
                     shift_output  ((i-1)*m  to  i*m  -  1)  <=
                                      shift_output(i*m - 1)  &
shift_output((i-
                                                  1)*m to i*m - 1 -
1);
            end loop;

            shift_shift <= not shift_shift;
            iteration <= iteration + 1;
            j <= j+1;

            if (j = m - 1) then
               if(i = 3) then
                 i <= 0;
                 iteration <= 0;
                 valid <= '1';
                 output_data <= input & output_t;
                 shift_en <= '0';
                 state <= STATE_Wait;
               else
                 state <= STATE_Load_mem;
                 i <= i + 1;
                 j <= 1;
```

```
              end if;
             end if;



            when STATE_Load_mem =>  -- 8 loads of shift memory
             if(i = 1) then
                for i in 1 to 4 loop
                     shift_output  ((i-1)*m  to  i*m  -  1)  <=
                              row17(i*m - 1) & row17((i-1)*m to
i*m
                                                                - 1 -
1);
              end loop;

              if(input(iteration) = '1') then
                 output_t <= output_t xor row17;
               end if;

             elsif(i = 2) then
                for i in 1 to 4 loop
                     shift_output  ((i-1)*m  to  i*m  -  1)  <=
                           row33(i*m - 1) & row33((i-1)*m to i*m

                                                                - 1 -
1);
              end loop;

              if(input(iteration) = '1') then
                 output_t <= output_t xor row33;
              end if;

             elsif(i = 3) then
                for i in 1 to 4 loop
                     shift_output  ((i-1)*m  to  i*m  -  1)  <=
                           row49(i*m - 1) & row49((i-1)*m to i*m
                                                                - 1 -
1);
              end loop;

              if(input(iteration) = '1') then
                 output_t <= output_t xor row49;
              end if;
             else
             end if;

             shift_load <= '1';
             shift_en <= '0';


             iteration <= iteration + 1;
             state <= STATE_Shifting;
```

```
            when others =>
             state <= STATE_Wait;
                end case;
          end if;
     end process;
end encoder_bh;
```

ДОДАТОК Б

Лістинг Б.1 – VHDL-модель алгоритму декодування

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library xil_defaultlib;
     use xil_defaultlib.records_pkg.all;

entity algorithm is
        Port ( counts : in STD_LOGIC_TH_ARRAY (0 to 128 - 1);
               clk : in STD_LOGIC;
               mem_set : std_logic;
               rst : std_logic;
               output : out STD_LOGIC_VECTOR (0 to 127));
    end algorithm;

    architecture Behavioral of algorithm is
                                              -----------     States
-------------------------------------------------------
        type STATES is (STATE_Wait, STATE_Processing);
        signal state: STATES := STATE_Wait;
         ------------------------------------------------------------
-------------------
        signal in_index : integer;
        signal in_value : std_logic_vector(0 to 2);
        signal it : integer;
        constant NumberOfVariableNodes : integer := 64;

    begin

        process(clk, rst)
        begin
            if(rst = '1') then
                state <= STATE_Wait;

            elsif(rising_edge(clk)) then
                case (state) is

                    when STATE_Wait =>
                        if (mem_set = '1') then
                            state <= STATE_Processing;
                            in_index <= 0;
                            in_value <= counts(0);
                            it <= 1;
                        end if;

                    when STATE_Processing =>
```

```
                        if (it = NumberOfVariableNodes) then
                                    if (in_value /= "000") then
-- ?????????????!!!
                                output <= (others => '0');
                                output(in_index) <= '1';
                            end if;
                            state <= STATE_Wait;

                        else
                             if ((counts(it)(2) > in_value(2)) or
-- ?????????????!!!
                                 (counts(it)(2) = in_value(2) and
counts(it)(1) > in_value(1)) or
                                 (counts(it)(2) = in_value(2) and
counts(it)(1)  =  in_value(1)  and  counts(it)(0)  >  in_value(0)))
then
                                    in_index <= it;
                                    in_value <= counts(it);
                                end if;
                            it <= it + 1;
                        end if;


                    when others =>
                         state <= STATE_Wait;
                    end case;
            end if;
        end process;

    end Behavioral;
```

Лістинг Б.2 – VHDL-модель модулю logic_module

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

library xil_defaultlib;
     use xil_defaultlib.records_pkg.all;
     use xil_defaultlib.f_files.all;


entity logic_module is
     port(
          decoded : in std_logic_vector(0 to 127);
          --------
          all_zero : out std_logic;
          counts : out STD_LOGIC_TH_ARRAY(0 to 127)
     );
end logic_module;
```

```
architecture logic_module of logic_module is
     constant ParityMatrixSizeM    : integer := 64;
     constant ParityMatrixSizeN    : integer := 128;
     signal syndrome : std_logic_vector(0 to ParityMatrixSizeM -
1);
     ----------------------------
     component counter_of_signals is
     port(
          a : in STD_LOGIC_VECTOR(5 downto 0);
           b : out STD_LOGIC_VECTOR(2 downto 0)
          );
     end component;
     ----------------------------
     component unary_inv_OR IS
    generic (
          N: integer := 64
     );
    port (
          inp: in std_logic_vector(N-1 downto 0);
        outp: out std_logic);
     end component;
     ----------------------------
     --constant path : string := "FBParity_4096.txt";
     --constant graph_inst : graph := my_ram_init(path);
begin

     Prepare_syndroms : for i in 0 to ParityMatrixSizeM - 1
generate
          P1:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size)        =        1        )        generate        --
graph_inst.child_node_array_inst(i).size = 1) generate
               syndrome(i)                                     <=
decoded(f_file_refactored(child_nodes + i*node + index0) - 1);
          end generate;

          P2:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 2) generate
               syndrome(i)                                     <=
decoded(f_file_refactored(child_nodes  +  i*node  +  index0)  -  1)
xor

decoded(f_file_refactored(child_nodes + i*node + index1) - 1);
          end generate;

          P3:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 3) generate
               syndrome(i)                                     <=
decoded(f_file_refactored(child_nodes  +  i*node  +  index0)  -  1)
xor

     decoded(f_file_refactored(child_nodes + i*node + index1) -
1) xor
```

```
     decoded(f_file_refactored(child_nodes + i*node + index2) -
1);
          end generate;

          P4:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 4) generate
               syndrome(i)                                    <=
decoded(f_file_refactored(child_nodes + i*node + index0) - 1)
xor

     decoded(f_file_refactored(child_nodes + i*node + index1) -
1) xor

     decoded(f_file_refactored(child_nodes + i*node + index2) -
1) xor

     decoded(f_file_refactored(child_nodes + i*node + index3) -
1);
          end generate;

          P5:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 5) generate
               syndrome(i)                                    <=
decoded(f_file_refactored(child_nodes + i*node + index0) - 1)
xor

     decoded(f_file_refactored(child_nodes + i*node + index1) -
1) xor

     decoded(f_file_refactored(child_nodes + i*node + index2) -
1) xor

     decoded(f_file_refactored(child_nodes + i*node + index3) -
1) xor

     decoded(f_file_refactored(child_nodes + i*node + index4) -
1);
          end generate;

          P6:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 6) generate
                                                  syndrome(i)    <=
decoded(f_file_refactored(child_nodes + i*node + index0) - 1)
xor
                              decoded(f_file_refactored(child_node
s + i*node + index1) - 1) xor
                              decoded(f_file_refactored(child_node
s + i*node + index2) - 1) xor
                              decoded(f_file_refactored(child_node
s + i*node + index3) - 1) xor
                              decoded(f_file_refactored(child_node
s + i*node + index4) - 1) xor
```

```
                                decoded(f_file_refactored(child_node
s + i*node + index5) - 1);
        end generate;

          P7:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 7) generate
                                                    syndrome(i)    <=
decoded(f_file_refactored(child_nodes  +  i*node  +  index0)  -  1)
xor
                                decoded(f_file_refactored(child_node
s + i*node + index1) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index2) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index3) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index4) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index5) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index6) - 1);
        end generate;

          P8:  if  (f_file_refactored(child_nodes  +  i*node  +
node_size) = 8) generate
                                                    syndrome(i)    <=
decoded(f_file_refactored(child_nodes  +  i*node  +  index0)  -  1)
xor
                                decoded(f_file_refactored(child_node
s + i*node + index1) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index2) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index3) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index4) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index5) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index6) - 1) xor
                                decoded(f_file_refactored(child_node
s + i*node + index7) - 1);
        end generate;
     end generate;

     Count_syndroms_failures: for i in 0 to ParityMatrixSizeN - 1
generate
          L1:  if  (f_file_refactored(parent_nodes  +  i*node  +
node_size) = 1) generate
               U0: counter_of_signals port map (
               a(0)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index0) - 1),
               a(5 downto 1) => (others => '0'),
```

```
                b => counts(i));
            end generate L1;


            L2:  if  (f_file_refactored(parent_nodes  +  i*node  +
node_size) = 2) generate
                U0: counter_of_signals port map (
                a(0)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index0) - 1),
                a(1)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index1) - 1),
                a(5 downto 2) => (others => '0'),
                b => counts(i));
            end generate L2;

            L3:  if  (f_file_refactored(parent_nodes  +  i*node  +
node_size) = 3) generate
                U0: counter_of_signals port map (
                a(0)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index0) - 1),
                a(1)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index1) - 1),
                a(2)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index2) - 1),
                a(5 downto 3) => (others => '0'),
                b => counts(i));
            end generate L3;

            L4:  if  (f_file_refactored(parent_nodes  +  i*node  +
node_size) = 4) generate
                U0: counter_of_signals port map (
                a(0)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index0) - 1),
                a(1)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index1) - 1),
                a(2)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index2) - 1),
                a(3)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index3) - 1),
                a(5 downto 4) => (others => '0'),
                b => counts(i));
            end generate L4;

            L5:  if  (f_file_refactored(parent_nodes  +  i*node  +
node_size) = 5) generate
                U0: counter_of_signals port map (
                a(0)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index0) - 1),
                a(1)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index1) - 1),
                a(2)  =>  syndrome(f_file_refactored(parent_nodes  +
i*node + index2) - 1),
                a(3)  =>  syndrome(f_file_refactored(parent_nodes  +
```

```
i*node + index3) - 1),
               a(4) => syndrome(f_file_refactored(parent_nodes +
i*node + index4) - 1),
               a(5) => '0',
               b => counts(i));
          end generate L5;

     end generate;


     all_zero_mark: unary_inv_OR generic map(ParityMatrixSizeM)
                         port map(syndrome, all_zero);


end logic_module;
```

Лістинг Б.3 – VHDL-модель декодера

```
-- m check nodes
-- n code/variable nodes


library IEEE;
     use IEEE.STD_LOGIC_1164.all;
     use ieee.numeric_std.all;
     use STD.textio.all;
     use ieee.std_logic_textio.all;

     -- Custom package with
library xil_defaultlib;
     use xil_defaultlib.records_pkg.all;


entity decoder is

        generic(
            InfowordLength          : integer := 64;        --
Length of infoword in codeword received from line
            CodewordLength          : integer := 128;        --
Length of codeword received from line
            MaxIterations             : integer := 50;
-- Maximum amount of decoding iterations
            NumberOfVariableNodes     : integer := 128;      --
Maximum of number of variable nodes
-- Change it in package!!!!
            NumberOfCheckNodes        : integer := 64         --
Maximum of number of check nodes
-- Change it in package!!!!
            );

        port(
```

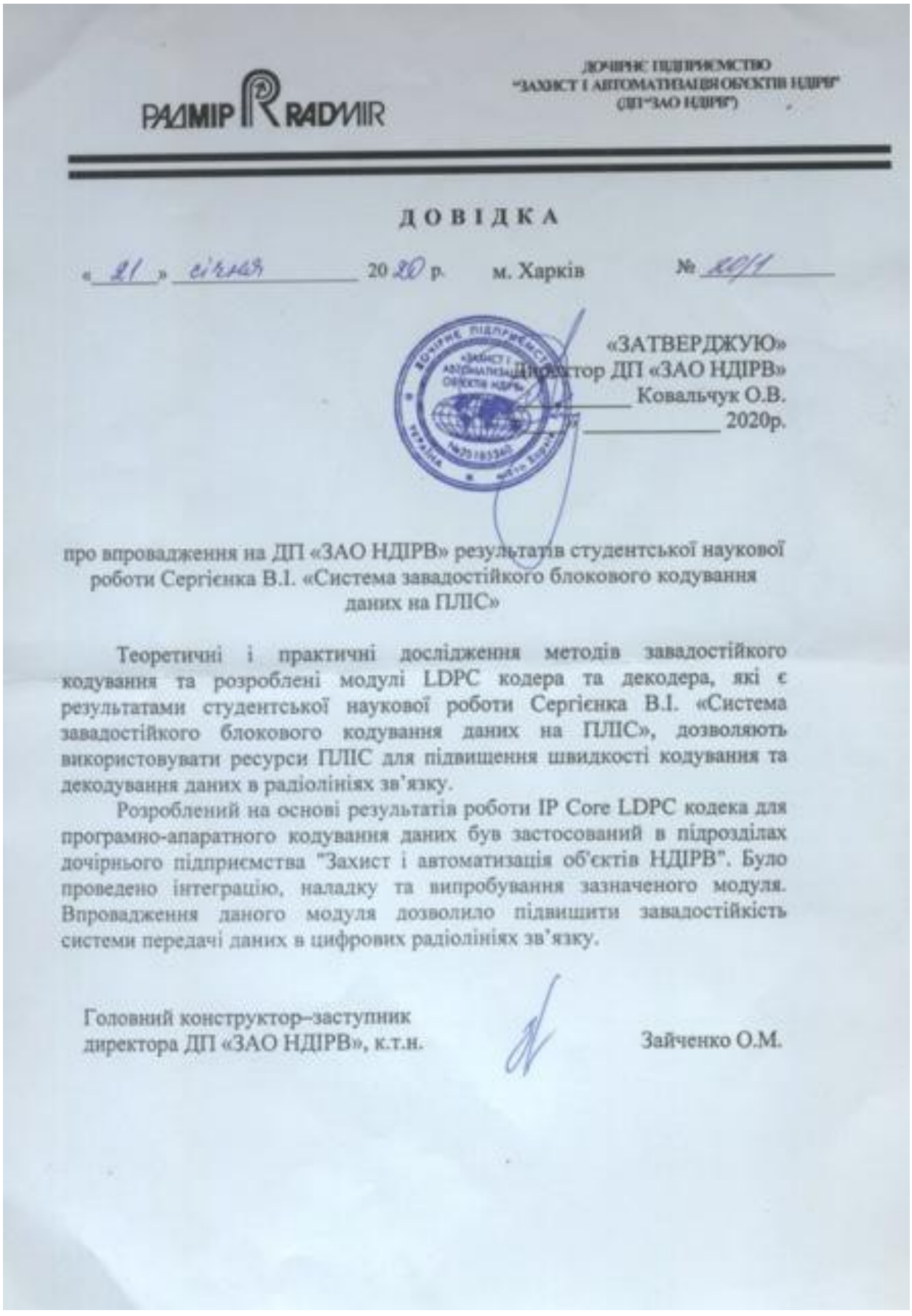
```
            input               : in std_logic_vector (0 to
CodewordLength - 1);
            input_clk            : in std_logic;
            clk_not_bfg            : in std_logic;
            rst                 : in std_logic;

------------------------------------------------------------
            output              : out std_logic_vector (0 to
InfowordLength - 1);
            IsValid             : out std_logic;
            waiting                : out std_logic;
            FinalNumIterations    : out integer
            );

    end decoder;


    architecture decoder_bh of decoder is
        ----------- Cmponents
-------------------------------------------------
        component logic_module is
        port(
            decoded : in std_logic_vector(0 to 128 - 1);
            --------
            all_zero : out std_logic;
            counts : out STD_LOGIC_TH_ARRAY(0 to 128 - 1)
        );
        end component logic_module;

------------------------------------------------------------------
-----------
        component BUFG
            port(I: in STD_LOGIC; O: out STD_LOGIC);
        end component;

------------------------------------------------------------------
-----------
        component decoded_memory is
            Port ( input : in STD_LOGIC_VECTOR (0 to 127);
                   set : in STD_LOGIC;
                   reset : in STD_LOGIC;
                   output : out STD_LOGIC_VECTOR (0 to 127));
        end component decoded_memory;

------------------------------------------------------------------
-----------
        component xor_module is
            Port ( xor1 : in STD_LOGIC_VECTOR (0 to 127);
                   xor2 : in STD_LOGIC_VECTOR (0 to 127);
                   xor_out : out STD_LOGIC_VECTOR (0 to 127));
        end component xor_module;
```

```
------------------------------------------------------------------
-----------
        component algorithm is
            Port (
                   counts : in STD_LOGIC_TH_ARRAY (0 to 128 -
1);
                   clk : in STD_LOGIC;
                   mem_set : std_logic;
                   rst : std_logic;
                   output : out STD_LOGIC_VECTOR (0 to 127));
        end component algorithm;

------------------------------------------------------------------
-----------

        ----------- States
-----------------------------------------------------
        type STATES is (STATE_Wait, STATE_Check, STATE_Dell);
        signal state: STATES := STATE_Wait;

------------------------------------------------------------------
-----------

        ----------- Signals for memory
-------------------------------------------
        signal decoded_input, decoded_output :
std_logic_vector(0 to 127);
        signal decoded_set, decoded_reset : std_logic;

------------------------------------------------------------------
-----------

        ----------- Internal temps
-------------------------------------------
        signal FinalNumIterations_tmp : integer := 0;

        signal counts : STD_LOGIC_TH_ARRAY (0 to 128 - 1);
        signal all_zero : std_logic := '0';
        signal clk : std_logic;
        signal xor1: std_logic_vector(0 to 127);
        signal new_decoded : std_logic_vector(0 to 127);
        signal dell_iter : integer := 0;
        signal algo_start : std_logic := '0';

------------------------------------------------------------------
-----------


    begin
        --buggmark: BUFG port map (clk_not_bfg, clk);
        clk <= clk_not_bfg;
```

```
        Logic_module_mark: logic_module port map
(decoded_output, all_zero, counts);
        Algo: algorithm port map (counts, clk,  algo_start, rst,
xor1);
        Xormodule: xor_module port map (xor1, decoded_output,
new_decoded);
        Memory: decoded_memory port map (input => decoded_input,
set => decoded_set, reset => rst, output => decoded_output);

        IsValid <= all_zero;
        waiting <= '1' when (state = STATE_Wait) else '0';

        process (clk, rst)
        begin
            if (rst = '1') then
            decoded_set <= '0';

            elsif (rising_edge(clk)) then
                case (state) is

                    when STATE_Wait =>
                        if (input_clk = '1') then
                            state <= STATE_Check;
                            decoded_input <= input;
                            decoded_set <= '1';
                        end if;


                    when STATE_Check =>

----------------------------------------
                        -- CONTINUE IF A CODEWORD HAS NOT BEEN
FOUND

----------------------------------------
                        decoded_set <= '0';
                        if (all_zero = '0') then
                                algo_start <= '1';
                                state <= STATE_Dell;
                            else
                                output <= decoded_output(0 to
63);
                                FinalNumIterations <=
FinalNumIterations_tmp;
                                state <= STATE_Wait;
                            end if;


                    when STATE_Dell =>
                        if (dell_iter = 65) then
                           dell_iter <= 0;
                           decoded_input <= new_decoded;
                           decoded_set <= '1';
```

```
                        algo_start <= '0';
                        state <= STATE_Check;
                    else
                        dell_iter <= dell_iter + 1;
                    end if;


                when others =>
                    state <= STATE_Wait;
            end case;
        end if;

    end process;
end decoder_bh;
```

ДОДАТОК В



ДОЧІРНЄ ПІДПРИЄМСТВО
"ЗАХИСТ І АВТОМАТИЗАЦІЯ ОБ'ЄКТІВ НДІРВ"
(ДП "ЗАО НДІРВ")

РАДМІР RADMIR

ДОВІДКА

«21» січня 2020 р. м. Харків № 2011

«ЗАТВЕРДЖУЮ»
Директор ДП «ЗАО НДІРВ»
_____ Ковальчук О.В.
_____ 2020р.

про впровадження на ДП «ЗАО НДІРВ» результатів студентської наукової роботи Сергієнка В.І. «Система завадостійкого блокового кодування даних на ПЛІС»

Теоретичні і практичні дослідження методів завадостійкого кодування та розроблені модулі LDPC кодера та декодера, які є результатами студентської наукової роботи Сергієнка В.І. «Система завадостійкого блокового кодування даних на ПЛІС», дозволяють використовувати ресурси ПЛІС для підвищення швидкості кодування та декодування даних в радіолініях зв'язку.

Розроблений на основі результатів роботи IP Core LDPC кодека для програмно-апаратного кодування даних був застосований в підрозділах дочірнього підприємства "Захист і автоматизація об'єктів НДІРВ". Було проведено інтеграцію, наладку та випробування зазначеного модуля. Впровадження даного модуля дозволило підвищити завадостійкість системи передачі даних в цифрових радіолініях зв'язку.

Головний конструктор–заступник
директора ДП «ЗАО НДІРВ», к.т.н. Зайченко О.М.

ДОДАТОК Г

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Автоматизації проектування обчислювальної техніки

Атестаційна робота

Моделі та методи проектування завадостійких систем бездротової передачі даних з використанням ПЛІС

Виконав:
студент гр. СКСм-19-1
Сергієнко Владислав

Керівник:
доц. каф АПОТ
Філіппенко І.В.

Актуальність тематики

2

Об'єкт і предмет дослідження

- **Об'єкт дослідження:** завадостійкі системи бездротової передачі даних.
- **Предмет дослідження:** методи побудови завадостійких систем бездротової передачі даних, шляхом застосування блокового кодування з використанням ПЛІС.

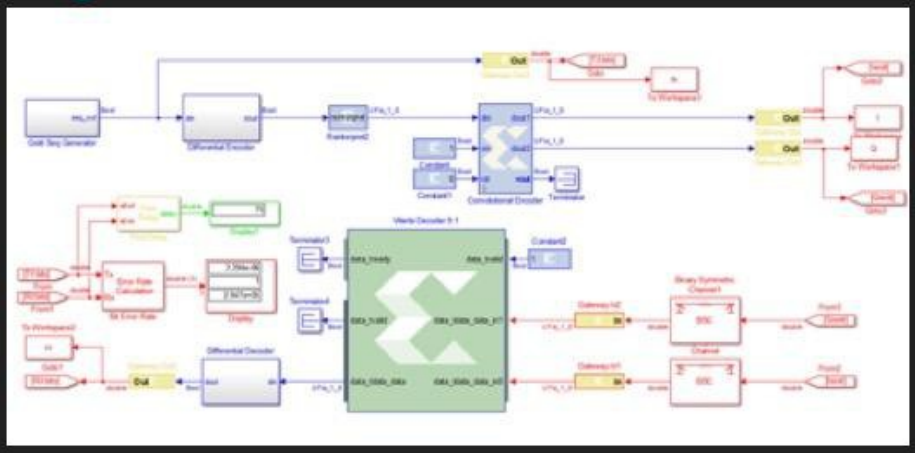
3

Мета дослідження і завдання

- **Мета дослідження:** розробити IP Core пристрою кодеку завадостійкої системи передачі даних на ПЛІС.

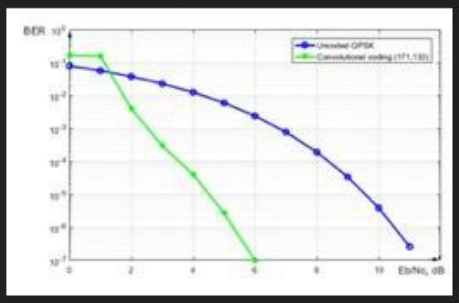
4

Результати роботи
(1) Математична модель радіолінії

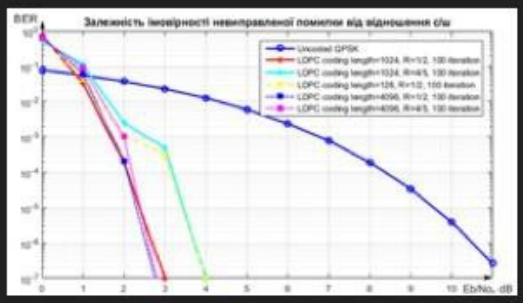


5

Результати роботи
(2) Порівняння графіків



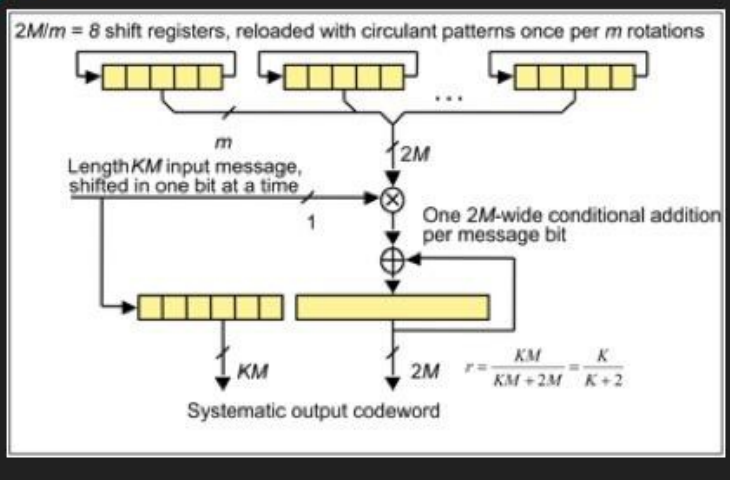
Графік залежності імовірності бітової помилки при згортковому кодуванні



Графік залежності імовірності бітової помилки при LDPC кодуванні

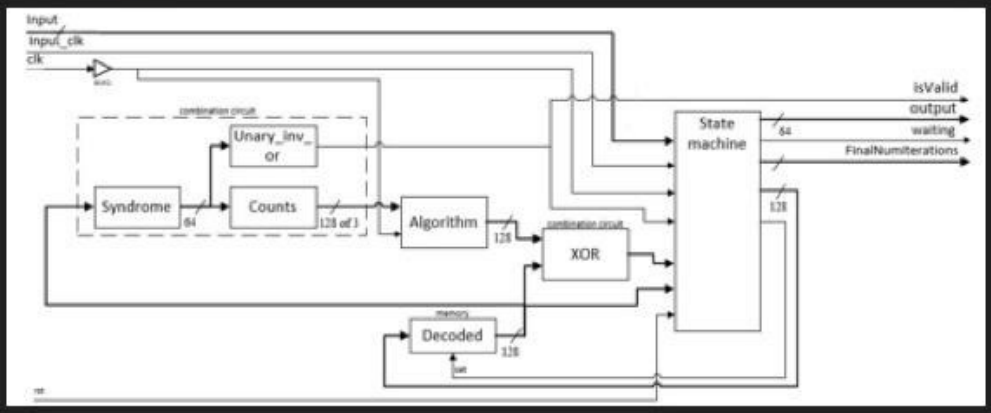
6

Результати роботи
(3) Алгоритм кодування даних



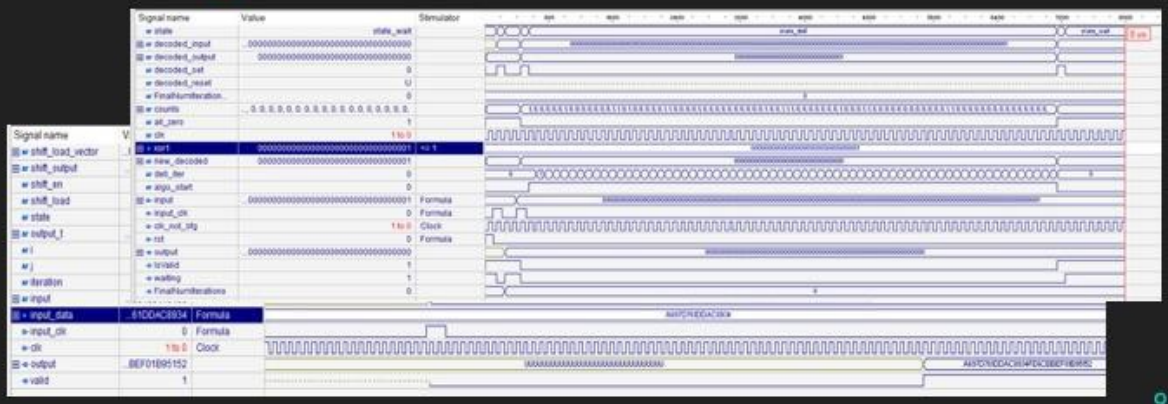
7

Результати роботи
(4) Схема розробленого декодера



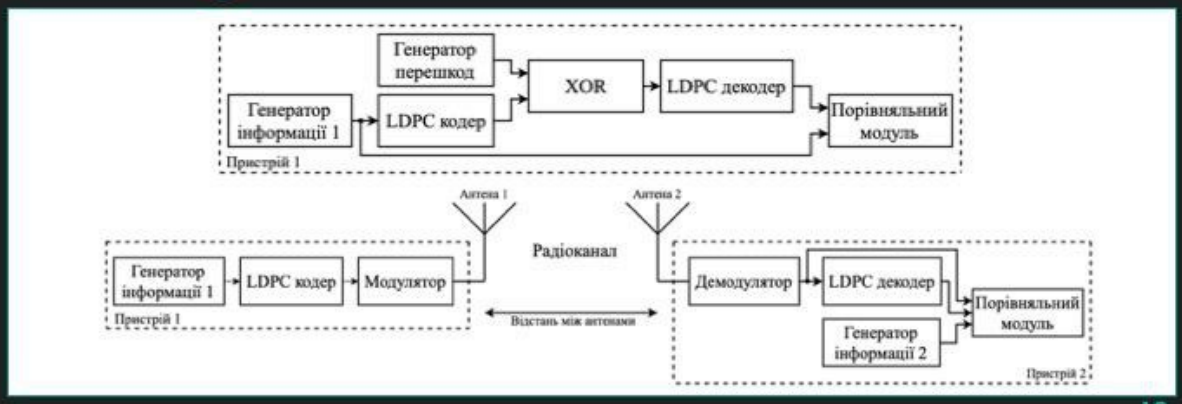
8

Результати роботи
(5) Процес веріфікації



9

Результати роботи
(6) Проведені діагностичні експерименти



10

Наукова новизна

- Отримано подальший розвиток алгоритму LDPC-кодування для реалізації його на ПЛІС, за рахунок використання зсувних регістрів, суматорів та керуючого автомата.
- Розроблено алгоритм та схеми LDPC-декодера на ПЛІС з використанням зсувних регістрів, суматорів, виконуючого автомата та керуючого автомата, який синхронізує роботу блоків пристрою.

11

Практична значимість

- Розроблені модулі можуть бути використані для реалізації завадостійкого LDPC кодування інформації в високошвидкісних системах передачі даних, а саме відеоданих із супутників, дронів, безпілотників. Результати роботи можна використовувати як готові модулі при розробці різноманітних пристроїв, в яких використовується завадостійке кодування, на приклад в радіолінії зв'язку з космічними апаратами. Це дає можливість зменшити часові витрати, підвищити якість проекту в цілому і автоматизувати процес проектування нових систем.

12

Публікація результатів роботи

Результати роботи опубліковано:

- Філіпенко І.В. Система завадостійкого кодування даних на ПЛІС / І.В. Філіпенко, Е.М. Кулак, В.І. Сергієнко // Радіоелектроніка та інформатика. – 2019. – № 4(87).– С. 44-51. (науковий журнал переліку ВАК)

- Сергієнко В.І. Реалізація алгоритму LDPC кодування за допомогою ПЛИС / В.І. Сергієнко // Тези доповіді 23-го Міжнародного молодіжного форуму «Радіоелектроніка та молодь у 21 столітті» Зб. матеріалів форуму Т.5 – Харків.: ХНУРЕ – 16-18 квітня 2019. с. 41-42.

13

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК.ХХХХХХ.018 ПЗ	Пояснювальна записка	65 с.
		Графічні матеріали	
2		Слайд-презентація	14 слайдів
		Носії інформації	
3		Компакт-диск (CD)	1 шт.

Змін.	Арк.	Номер докум.	Підп.	Дата	ГЮІК.ХХХХХХ.018 ВД			
Розроб.		Сергієнко В.І.			Моделі та методи проектування завадостійких систем бездротової передачі даних з використанням ПЛІС Відомість атестаційної роботи	Літ	Аркуш	Аркушів
Перевір.		Філіппенко І.В.				У		
Н.контр.		Рожнова Т.Г.				ХНУРЕ Кафедра АПОТ		
Затв.		Чумаченко С.В.						